CSC110 Assignment 4: Number Theory, Cryptography, and Algorithm Running Time

Jamie Yi

November 24, 2021

Part 1: Practice with Proofs

1. Statement to prove:

$$\forall a, k, n \in \mathbb{Z},\ \ \gcd(a, n) = 1 \Rightarrow \gcd(a + kn, n) = 1$$

Proof. :
-Let $e \in \mathbb{N}$
-Asusume $e|a + kn$, this implies: $\exists k_1 \in \mathbb{Z}$ s.t. $a + kn = k_1 e$
-Asusume $e|n$, this implies: $\exists k_2 \in \mathbb{Z}$ s.t. $n = k_2 e$

-We can rearrange $a + kn = k_1 e$ to be $a = k_1 e - kn$
-Substitute in $n = k_2 e$, to get $a = k_1 e - k(k_2 e) = e(k_1 - kk_2)$
-Thus, $e|a$

-Since $e|n$(by assumption) and $e|a$ and $\gcd(a, n) = 1$, $e \leq 1$
-Since $e = \gcd(a + kn, n)$ and $e \leq 1$ and 1 divides all numbers, therefore $\gcd(a + kn, n) = 1$ as needed

□

2. Statement to prove (we've expanded the definition of Omega for you!):

$$\exists c, n_0 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_0 \Rightarrow \log_3 n - \log_{11} n \geq c \cdot \log_{14} n$$

Proof. :
-let $c = 1.2$
-Let $n_0 = 1$
-We know $2.4 < \frac{1}{\log_{14} 3} < 2.5$ and $1.1 < \frac{1}{\log_{14} 3} < 1.2$

-Since $n \geq 1$, $\log_{14} n$, $\log_3 n$, and $\log_{11} n$ will never be negative
$log_3 n - log_{11} n$
$= \frac{\log_{14} n}{\log_{14} 3} - \frac{\log_{14} n}{\log_{14} 11}$
$= \log_{14} n \frac{1}{\log_{14} 3} - \log_{14} n \frac{1}{\log_{14} 11}$
$= \log_{14} n \frac{1}{\log_{14} 3} + (-\log_{14} n \frac{1}{\log_{14} 11})$
$\geq 2.4 \log_{14} c + (-1.2 \log_{14} n)$
$= 1.2 \log_{14} n$

-Therefore $\log_3 n - \log_{11} n \geq c \cdot \log_{14} n$ as needed

□

3. Statement to prove (we haven't expanded the definition of Big-O for you, but we encourage you to do so yourself):

$$\forall f, g : \mathbb{N} \to \mathbb{R}^{\geq 0},\ g \in \mathcal{O}(f) \wedge \big(\forall m \in \mathbb{N},\ f(m) \geq 1\big) \Rightarrow g \in \mathcal{O}(\lfloor f \rfloor)$$

Proof. Definition of $g \in \mathcal{O}(f)$ (our assumption):

$$\exists c_1, n_1 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_1 \Rightarrow g(n) \leq c_1 \cdot f(n)$$

Definition of $g \in \mathcal{O}(\lfloor f \rfloor)$ what we WTS:

$$\exists c_2, n_2 \in \mathbb{R}^+,\ \forall n \in \mathbb{N},\ n \geq n_2 \Rightarrow g(n) \leq c_2 \cdot \lfloor f(n) \rfloor$$

-Take $c_2 = 2c_1$
-Take $n_2 = n_1$
-Using what we know that $x < \lfloor n \rfloor + 1$, we know that $c_1 \cdot n < c_1 \cdot \lfloor n \rfloor + c_1$
-We also know that $\lfloor m \rfloor \geq 1$(for all $m \in \mathbb{N}$)

$g(n) \leq c_1 \cdot f(n)$ (by assumption)
$< c_1 \cdot \lfloor f(n) \rfloor + c_1$
$\leq c_1 \cdot \lfloor f(n) \rfloor + c_1 \cdot \lfloor f(n) \rfloor$ (The inequality holds since we know $\lfloor f(x) \rfloor \geq 1$)
$= 2c_1 \cdot \lfloor f(n) \rfloor$
$= c_2 \cdot \lfloor f(n) \rfloor$ as needed

$\square$

Part 2: Generating Coprime Numbers

1. Not to be handed in.
2. Complete this part in the provided `a4_part2.py` starter file. Do **not** include your solution in this file.
3. Prove that each loop invariant holds.

   a. Loop Invariant 1

   Proof. :
   -Let i_0 be `len(nums_so_far) - 1` at the start of the iteration
   -Take $i = i_0 + 1$

   We know at the start of the iteration, for all k in `nums_so_far`, $\gcd(k, 2) = 1$ and $\gcd(k, 3) = 1$.
   Looking at the first line in the while, we can also tell that k can be written as $k = 1 + 6e$ or $k = 5 + 6e$ where e is some $\mathbb{Z}$ (e does not need to be the same for every instance of k)

   Case 1: `nums_so_far[`i_0`]` $= 1 + 6e$, where $e \in \mathbb{Z}$
   -In this case, at the end of the iteration, `nums_so_far[`i`]` $= 5 + 6e$
   -We know $\gcd(5, 2) = 1$, so by Part 1 Question 1 we know $\gcd(5 + 2(3e), 2) = 1$
   -We know $\gcd(5, 3) = 1$, so by Part 1 Question 1 we know $\gcd(5 + 3(2e), 3) = 1$

   Case 2: `nums_so_far[`i_0`]` $= 5 + 6e$, where $e \in \mathbb{Z}$
   -In this case, at the end of the iteration, `nums_so_far[`i`]` $= 7 + 6e$, or $= 1 + 6(e + 1)$
   -We know $\gcd(1, 2) = 1$, so by Part 1 Question 1 we know $\gcd(1 + 2(3 \cdot (e + 1), 2) = 1$
   -We know $\gcd(1, 3) = 1$, so by Part 1 Question 1 we know $\gcd(1 + 3(2 \cdot (e + 1), 3) = 1$

   Therefore at the end of the iteration, every number k in `nums_so_far` is coprime to 2 and coprime to 3. $\square$

   b. Loop Invariant 2

   Proof. :
   -Let i_0 be `len(nums_so_far)` $- 3$ at the start of the iteration
   -Take $i = i_0 + 1$

   We know for all j between 0 and `len(nums_so_far)` $- 2$, `nums_so_far[`i_0`]` $+ 6 =$ `nums_so_far[`$i_0 + 2$`]`.
   The first two lines of the iteration, we're appedning the value `nums_so_far[`$i_0 + 1$`]` $+ 6$ to the end of the list (`nums_so_far[`$i_0 + 1$`]` is the same index as `nums_so_far[`-2`]`, they both refer to the second last element of the list). The value is appended to index `nums_so_far[`$i_0 + 3$`]` (increasing the length of the list).

   Thus, `nums_so_far[`$i_0 + 1$`]` $+ 6 =$ `nums_so_far[`$i_0 + 3$`]`
   This can be rewritten as `nums_so_far[`i`]` $+ 6 =$ `nums_so_far[`$i + 2$`]`
   Now that the length of the list has increased, i now represents `len(nums_so_far) - 3`

   Therefore, for all natural numbers j between 0 and `len(nums_so_far)` $- 3$ inclusive,
   `nums_so_far[`i`]` $+ 6 =$ `nums_so_far[`$i + 2$`]` by the end of the iteration. $\square$

   c. Loop Invariant 3

Proof. :
-Let i_0 be `len(nums_so_far)` $-\ 2$ at the start of the iteration
-Take $i = i_0 + 1$

We know `nums_so_far[`i_0`]` $<$ `nums_so_far[`$i_0 + 1$`]` at the start of the iteration. By loop invariant 2, we know the value `nums_so_far[`i_0`]` $+\ 6$ will be appended to the list at index `nums_so_far[`$i_0 + 2$`]`. So far, we know that `nums_so_far[`i_0`]` $+\ 6 =$ `nums_so_far[`$i_0 + 2$`]`.
Seeing as how `nums_so_far` is initially $[1, 5]$, we can deduce that for any two consecutive elements in the list, `nums_so_far[`$j + 1$`]` - `nums_so_far[`j`]` ≤ 4. We can deduce this knowing that the difference between 5 and 1 is 4, and we're only adding to previous lists by a constant 6, and that the difference between the constant 6 and the initial difference 4 is 2. Thus consecutive elements will always be increasing, and either have a difference of 4 or 2.
From this we can say that `nums_so_far[`i_0`]` $<$ `nums_so_far[`$i_0 + 1$`]` $\leq$ `nums_so_far[`i_0`]` $+\ 4$
This implies `nums_so_far[`i_0`]` $<$ `nums_so_far[`$i_0 + 1$`]` $<$ `nums_so_far[`i_0`]` $+\ 6$
Which can be simplified as `nums_so_far[`$i_0 + 1$`]` $<$ `nums_so_far[`$i_0 + 2$`]` (by loop invariant 2)
Which can be rewritten as `nums_so_far[`i`]` $<$ `nums_so_far[`$i + 1$`]`

Therefore, for all natural numbers j between 0 and `len(nums_so_far)` $-\ 2$ inclusive, `len(nums_so_far[`j`])` $<$ `len(nums_so_far[`$j + 1$`])` (this means that `nums_so_far` is always sorted). $\square$

d. Loop Invariant 4

Proof. :
-Let i_0 be `len(nums_so_far)` $-\ 1$ at the start of the iteration
-Take $i = i_0 + 1$

We know at the start of the iteration, for all natural numbers k in between 0 and `len(nums_so_far[`i_0`])` inclusive, if k is coprime to 2 and coprime to 3, then k in `len(nums_so_far)`.
By loop invariant 1, we know k can be written as $k = 1 + 6e$ or $k = 5 + 6e$ where e is some $\mathbb{Z}$ (e does not need to be the same for every instance of k).

Case 1: `nums_so_far[`i_0`]` $= 1 + 6e$, where $e \in \mathbb{Z}$
-In this case, by loop invariant 1 we know at the end of the iteration, `nums_so_far[`i`]` $= 5 + 6e$
-We must verify that for all natural numbers k in between $2 + 6e$ and $5 + 6e$ inclusive, the implication $\gcd(k, 2) = 1 \wedge \gcd(k, 3) = 1 \implies k \in$ `nums_so_far` must hold
-We know that of the numbers in the range above, only $5 + 6e$ is in `nums_so_far`
$1 + 6e + 1 = 2 + 6e = 2(1 + 3e)$ thus, $2|2 + 6e$ and $2|2$ and so $\gcd(2 + 6e, 2) \neq 1$
$1 + 6e + 2 = 3 + 6e = 3(1 + 2e)$ thus, $3|3 + 6e$ and $3|3$ and so $\gcd(3 + 6e, 3) \neq 1$
$1 + 6e + 3 = 4 + 6e = 2(2 + 3e)$ thus, $2|4 + 6e$ and $2|2$ and so $\gcd(4 + 6e, 2) \neq 1$
$1 + 6e + 4 = 5 + 6e$ by loop invariant 1, we know $gcd(5 + 6e, 2) = 1 \wedge gcd(5 + 6e, 3) = 1$
$5 + 6e$ is the only number of the above in `nums_so_far`, thus satisfying the inequality for all natural numbers k in between 0 and `len(nums_so_far[`i`])` inclusive by the end of the iteration

Case 2: `nums_so_far[`i_0`]` $= 5 + 6e$, where $e \in \mathbb{Z}$
-In this case, by loop invariant 1 we know at the end of the iteration, `nums_so_far[`i`]` $= 1 + 6(e + 1)$
-We must verify that for all natural numbers k in between $6+6e$ and $1+6(e+1)$ inclusive, the implication $\gcd(k, 2) = 1 \wedge \gcd(k, 3) = 1 \implies k \in$ `nums_so_far` must hold
-We know that of the numbers in the range above, only $1 + 6(e + 1)$ is in `nums_so_far`
$5 + 6e + 1 = 6 + 6e = 2(3 + 3e)$ thus, $2|6 + 6e$ and $2|2$ and so $\gcd(6 + 6e, 2) \neq 1$
$5+6e+2 = 7+6e = 1+6(e+1)$ by loop invariant 1, we know $gcd(1+6(e+1), 2) = 1 \wedge gcd(1+6(e+1), 3) = 1$
$1 + 6(e + 1)$ is the only number of the above in `nums_so_far`, thus satisfying the inequality for all natural numbers k in between 0 and `len(nums_so_far[`i`])` inclusive by the end of the iteration

Therefore at the end of the iteration, for all natural numbers k in between 0 and `len(nums_so_far[`i`])` inclusive, if k is coprime to 2 and coprime to 3, then k in `len(nums_so_far)`. $\square$

4. Complete this part in the provided `a4_part2.py` starter file. Do **not** include your solution in this file.
5. Complete this part in the provided `a4_part2.py` starter file. Do **not** include your solution in this file.

Part 3: Running-Time Analysis

1. -Let n be the input integer n
   -Let k be the number of iterations the loop has ran
   -Let i_k be the value of `nums_so_far[`i`]` + 6 on the k-th iteration

   The cost of the assignment statement at the start is constant time, it takes 1 step.

   To analyze the while loop, we need to determine the cost of each iteration and the total number of iterations.
   -Each iteration has 2 constant time statements, so we'll count that as one step.
   -To find the number of iterations, we need to find the smallest value of k such that $i_k \geq n$ (making the loop condition false). There are two possible formulas for i_k:
   if k is odd, $i_k = 1 + 6(\frac{k+1}{2})$
   if k is even, $i_k = 5 + 6(\frac{k}{2})$
   -To find how many times the loop iterates at most, use the formula that will always be less than the other, thus requiring a greater k value to satisfy the inequality (we want to find the greater upper bound). In this case, it's the first formula. Then isolate for k in the inequality $i_k \geq n$
   $i_k \geq n$
   $1 + 6(\frac{k+1}{2}) \geq n$
   $\frac{k+1}{2} \geq \frac{n-1}{6}$
   $k \geq \frac{n-1}{3} - 1$
   -We need to find the smallest value of k such that $k \geq \frac{n-1}{3} - 1$, which is the definition of a ceiling function, and so the smallest value of k can be expressed as $k \geq \lceil \frac{n-1}{3} - 1 \rceil$ We know the loop runs at most $\lceil \frac{n-1}{3} - 1 \rceil$ times, with one step per iteration, for a total of $\lceil \frac{n-1}{3} - 1 \rceil$ steps.

   The cost of the return statement is constant time, it takes 1 step.

   Putting it all together, the function `coprime_to_2_and_3` has a running time of
   $1 + \lceil \frac{n-1}{3} - 1 \rceil + 1 = \lceil \frac{n-1}{3} - 1 \rceil + 2$ which is $\mathcal{O}(n)$

2. -Let P be the size of the input set `primes`
   -Let m be the product of the numbers in `primes`

   The cost of the first assignment statement is constant time, it takes 1 step.

   The second assignment statement have a running time of n steps, where n is the size of the input. So in this case, it takes P steps.

   To analyze the loop, we will first determine the number of steps in the inner loop and then the number of steps in the outer loop.
   -The inner loop contains one constant time statement, so we'll count that as one step per iteration of the inner loop.
   -The inner loop iterates for each element in `primes`, so it iterates P times with 1 step per iteration.
   -The outer loop contains 2 constant time statements and the inner loop that iterates P times, so we'll count it as taking $P + 1$ steps.
   -The outer loop runs one less time than the product of the numbers in `primes`, so it iterates $m - 1$ times, with $P + 1$ steps per iteration; for a total of $(m - 1) \cdot (P + 1)$.

The cost of the return statement is constant time, it takes 1 step.

Putting it all together, the function `starting_coprime_numbers` has a running time of
$1 + P + (m-1)\cdot(P+1) + 1 = (m-1)\cdot(P+1) + P + 2$ which is $\Theta(m \cdot P)$

3. -Let P be the size of the input set `primes`
-Let m be the product of the numbers in `primes`
-Let n be the input integer n
-Let k be the number of iterations the loop has ran
-Let i_k be the value of `nums_so_far[`i`]` + 6 on the k-th iteration

The first assignment statement is a function call to `starting_coprime_numbers`, which as established previously, takes $(m-1)\cdot(P+1)+P+2$ steps.

The second assignment statement is constant time, it takes 1 step.

The third assignment statement have a running time of n steps, where n is the size of the input. So in this case, it takes P steps.

To analyze the while loop, we need to determine the cost of each iteration and the total number of iterations.
-Each iteration has 2 constant time statements, so we'll count that as one step.
-To find the number of iterations, we need to find the smallest value of k such that $i_k \geq n$ (making the loop condition false). Using the same thought process from question 1 to find the formula that gives us the greatest k-value to find the greater upper bound between all the possible formulas. The formula is this case is:
$1 + m(1 + \lfloor \frac{k}{\phi(m)} \rfloor)$
-ϕ here is representing Euler's Totient Function -To find how many times the loop iterates at most, isolate for k in the inequality $i_k \geq n$
$i_k \geq n$
$1 + m(1 + \lfloor \frac{k}{\phi(m)} \rfloor) \geq n$
$1 + \lfloor \frac{k}{\phi(m)} \rfloor \geq \frac{n-1}{m}$
$\frac{k}{\phi(m)} \geq \lceil \frac{n-1}{m} + 1 \rceil$
$k \geq \phi(m) \lceil \frac{n-1}{m} + 1 \rceil$
We know the loop runs at most $\phi(m)\lceil \frac{n-1}{m} + 1 \rceil$ times, with one step per iteration, for a total of $\phi(m)\lceil \frac{n-1}{m} + 1 \rceil$ steps.

The cost of the return statement is constant time, it takes 1 step.

Putting it all together, the function `coprime_to_all` has a running time of
$(m-1)\cdot(P+1) + P + 2 + 1 + P + \phi(m)\lceil \frac{n-1}{m} + 1 \rceil + 1 = (m-1)\cdot(P+1) + \phi(m)\lceil \frac{n-1}{m} + 1 \rceil + 2P + 4$
which is $\mathcal{O}(m \cdot P + \frac{n}{m})$

Part 4: Two New Cryptosystems

Complete this part in the provided `a4_part4.py` starter file. Do **not** include your solution in this file.